Design of Dynamic Web Systems

Peder Johansson pedjoh-7@student.ltu.se Chonratid Pangdee chopan-7@student.ltu.se

January 2021

Assignment introduction

In this project we took on the role as a web developing company who had been tasked to create a web system for controlling the production and consummation of electricity for several households in a small scaled market. We were also asked to create a simulator that was supposed to simulate wind as well as the electricity that was produced and consumed by users.

System introduction

GraphQL

GraphQL is a opensource query and manipulation language that is used for APIs. In the project we have used GraphQL to develop APIs for getting and sending data between our frontend and backend.

nodejs

Nodejs is a runtime environment that enables running JavaScript application from the serverside. We use it to run our simulator.

SQLite

SQLite is a relations database management system. We used it for as our database to store all needed data for this project.

Libraries

During the project we have used many different libraries the following are the ones that have had the most importance.

- **React**
- **Bootstrap**
- **Axios**
- **JSON web token (JWT)**

React is a javascript library which simplify the creation of interactive and reusable user interfaces and UI components. React is the library we have used the most. We created our whole frontend using React.

Bootstrap is one of the most popular framework for web design. We have used bootstrap to get a better looking and more user friendly frontend design.

Axios is a Javascripit library that is used for creating http requests. We have used axios for creating http request to our APIs in the frontend of our program.

JWT is library used for creating access tokens to certify user identity. We have used JWT to create access tokens for users to verify that they have access to certain data.

Design choices

When creating the project we used modular programming. We wanted to create the different modules first and later on linking them together. Creating the project using this method has lots of benefits. It allow project members to work independently of each other. This is great for speeding up development since group members can put a bigger focus on working on parts were they have previous knowledge. When thinking of the learning aspect this method have had the downside of not letting members gain as much new knowledge as possible. Using modular programming simplifies further development since the different parts of the program are not heavily dependent on

each other. This means that it becomes easier to update certain parts of the project with out ruining the functionality of the other parts.

When working on the individual parts of the code we had the idea of loose coupling in mind. We wanted different parts work independently, the reasoning was basically the same as for why we used modular programming. We wanted it to be easy to update code without loosing functionality in other functions or classes.

Scalability analysis

At the begging of the project we were choosing between using graphQL or REST in the end we decided to use graphQL. GraphQL is able to retrieve data more efficiently than REST since graphQL enables us to fetch exactly the data we want with a single request, REST tend to have the problem that responses often have to much or to little data. Another advantage of grahphQL is that we are able to collect data from multiple sources using a single request. A disadvantage with graphQL is that it's hard to handle complex query. We did not encounter this problem, but it could become a problem if someone were to expand on this project and add more complexity to the queries.

We believe we may face issues regarding scalability because of the usage of SQLite since it is not designed for scalability. We would most likely get issues if we were to get a lot of traffic on our web application. To improve scalibaility one idea would be to change database to a more scaleable one.

Security analysis

One of the main security features we implemented is access tokens that we save in cookies. Users get a access token in order for the server to verify the user and only giving that user information that they are meant to have access to. We also implemented password encryption to strengthen the security for the users. One security flaw we have is that if someone were to get access to a users access token while they are still logged in that person would be able to gain access to the users information.

Challenges

A lot of the challenges we faced during the project was regarding lack of knowledge. We were both new to graphQL, JavaScript and React this lead to a lot of minor problems leading to many things taking more time than we initially thought they would take. The backend is able the handle more than we were able to implement in the frontend. We underestimated the time it would take to get frontend working as intended. The main cause for problems has been time management and estimating how long it would take to do certain implementations. This is quite unfortunate since we had intended to implement more features than the basic ones that were required but in the end were not able to due to time restraints.

Future work

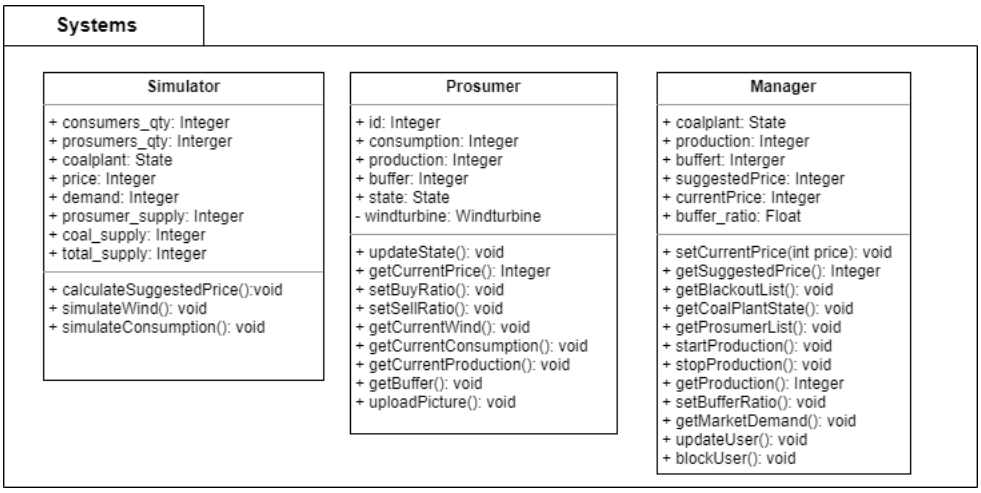
Some future work that would not be hard to implement is other weather features than wind. We have a json file containing lot of data of different weather features. We used real wind data to create random wind data for our simulator (the reason being to create realistic weather conditions). This can be done with the other weather data we have. The functions we use for the wind could quite easily be changed to be able to handle other types of information. If we had more time we would make a cleaner design for our frontend. We waited until the end to create our frontend, and because of the lack of time we had left until the deadline we had to prioritise getting the wanted functionality over a better looking design.

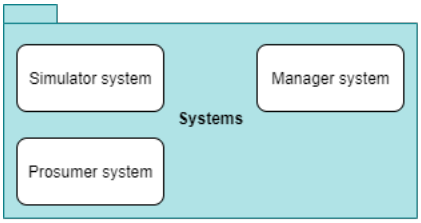
References

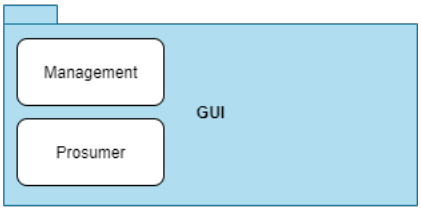
https://www.smhi.se/data/meteorologi/vind
https://reactjs.org/tutorial/tutorial.html
https://www.npmjs.com/package/axios

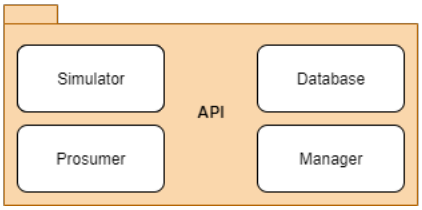
https://jwt.io/introduction/
https://www.w3schools.com/js/DEFAULT.asp
https://react-bootstrap.github.io/getting-started/introduction/

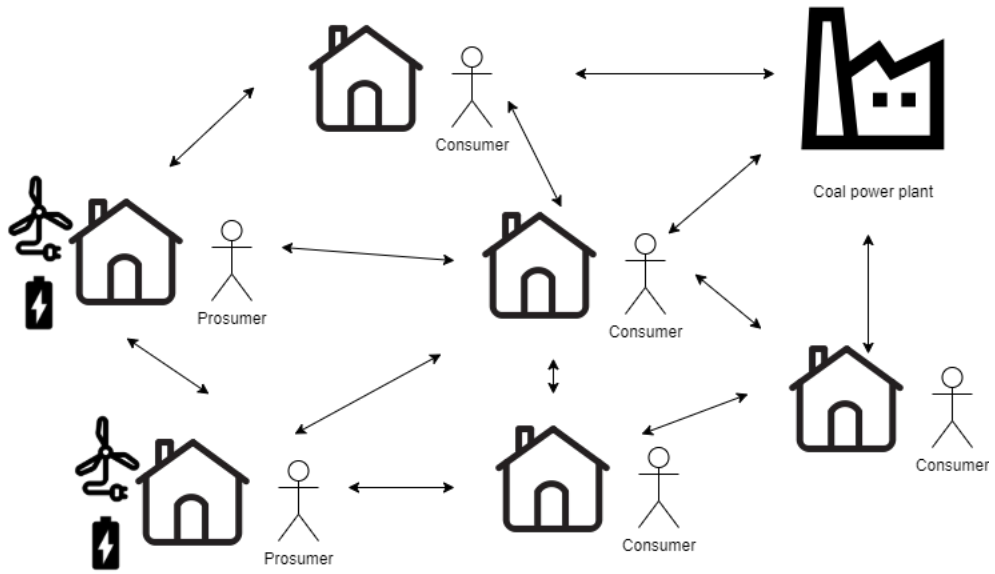
Napkinsketches

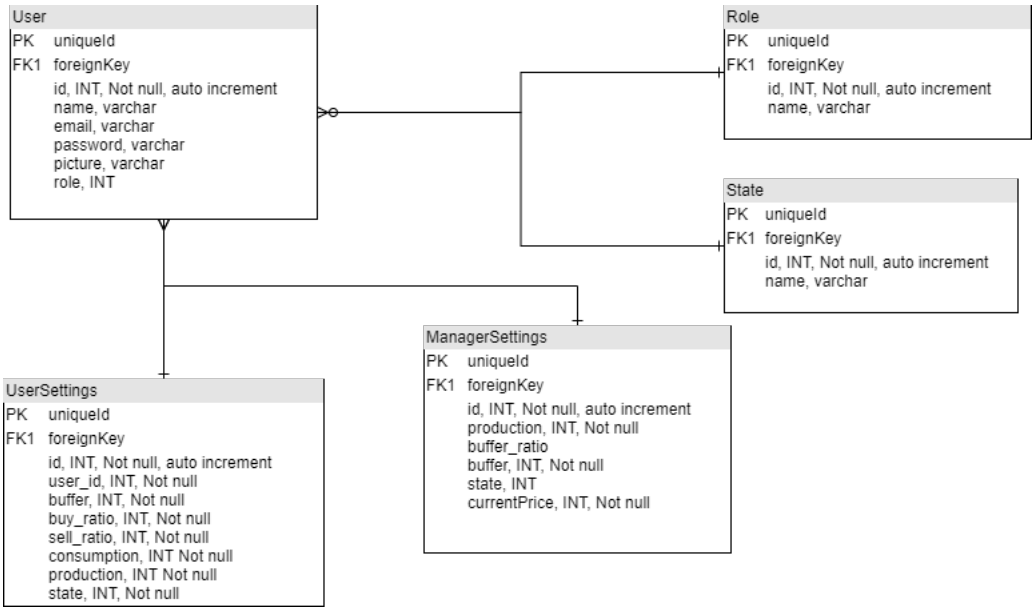


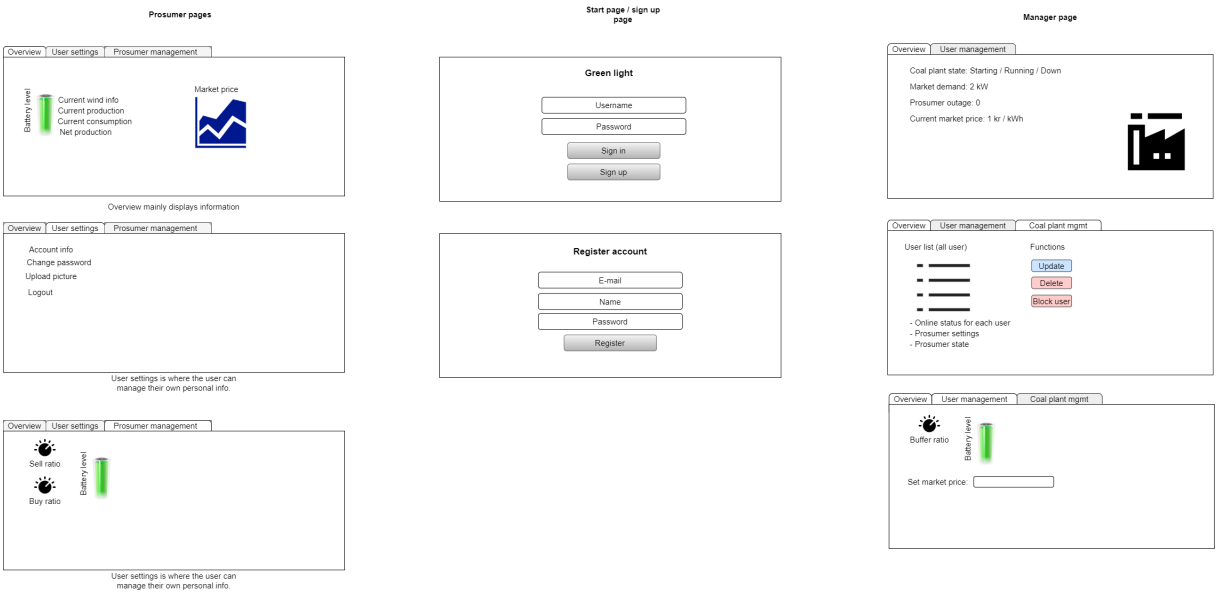









User
PK uniqueId
FK1 foreignKey
id, INT, Not null, auto increment
name, varchar
email, varchar
password, varchar
picture, varchar
role, INT
Role
PK uniqueId
FK1 foreignKey
id, INT, Not null, auto increment
name, varchar
State
PK uniqueId
FK1 foreignKey
id, INT, Not null, auto increment
name, varchar
ManagerSettings
PK uniqueId
FK1 foreignKey
id, INT, Not null, auto increment
production, INT, Not null
buffer_ratio
buffer, INT, Not null
state, INT
currentPrice, INT, Not null
UserSettings
PK uniqueId
FK1 foreignKey
id, INT, Not null, auto increment
user_id, INT, Not null
buffer, INT, Not null
buy_ratio, INT, Not null
sell_ratio, INT, Not null
consumption, INT Not null
production, INT Not null
state, INT, Not null


Prosumer pages
Overview
User settings
Prosumer management
Battery level
Current wind info
Current production
Current consumption
Net production
Market price
Overview mainly displays information
Account info
Change password
Upload picture
Logout
User settings is where the user can manage their own personal info.
Sell ratio
Buy ratio
Battery level
User settings is where the user can manage their own personal info.
Start page / sign up page
Green light
Username
Password
Sign in
Sign up
Register account
E-mail
Name
Password
Register
Manager page
Overview
User management
Coal plant state: Starting / Running / Down
Market demand: 2 kW
Prosumer outage: 0
Current market price: 1 kr / kWh
Coal plant mgmt
User list (all user)
Functions
Update
Delete
Block user
- Online status for each user
- Prosumer settings
- Prosumer state
Buffer ratio
Battery level
Set market price: